Assignment-12.5

Name:M.Varsha

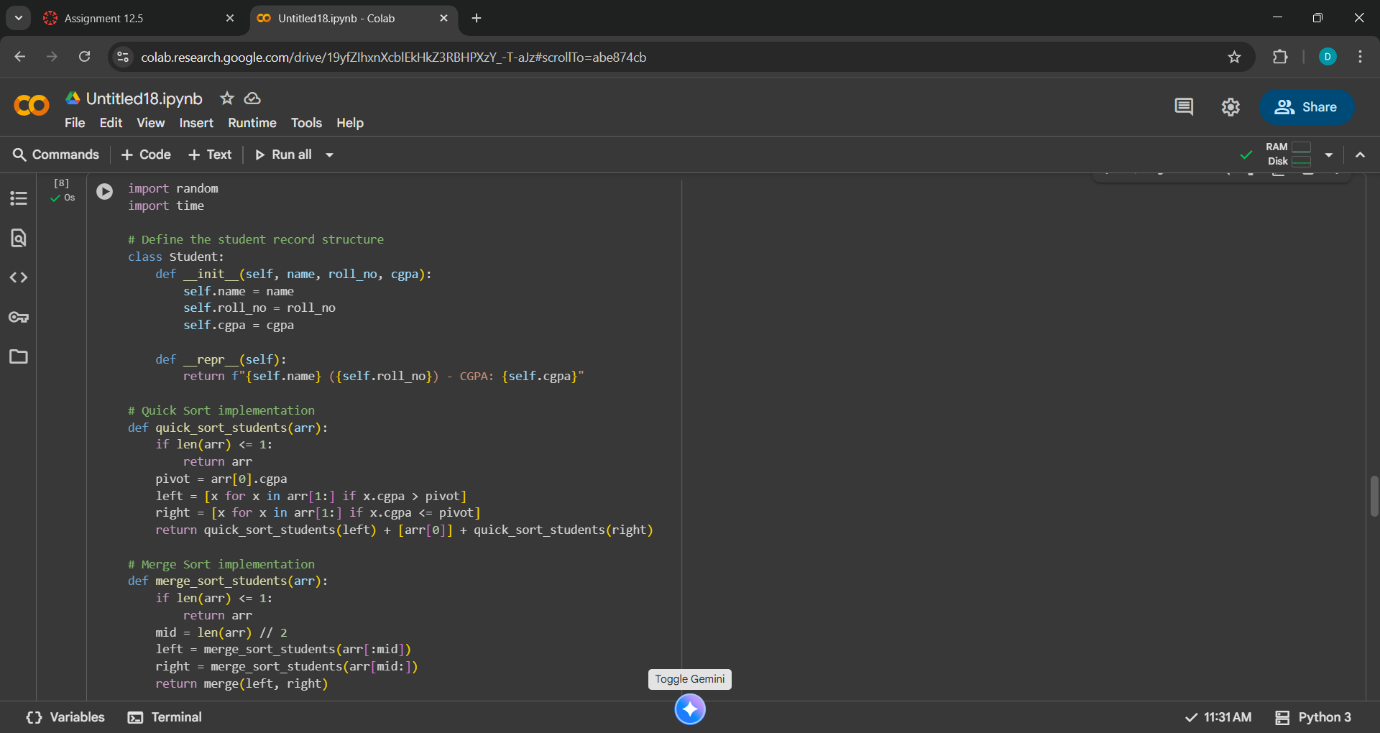
Batch:13

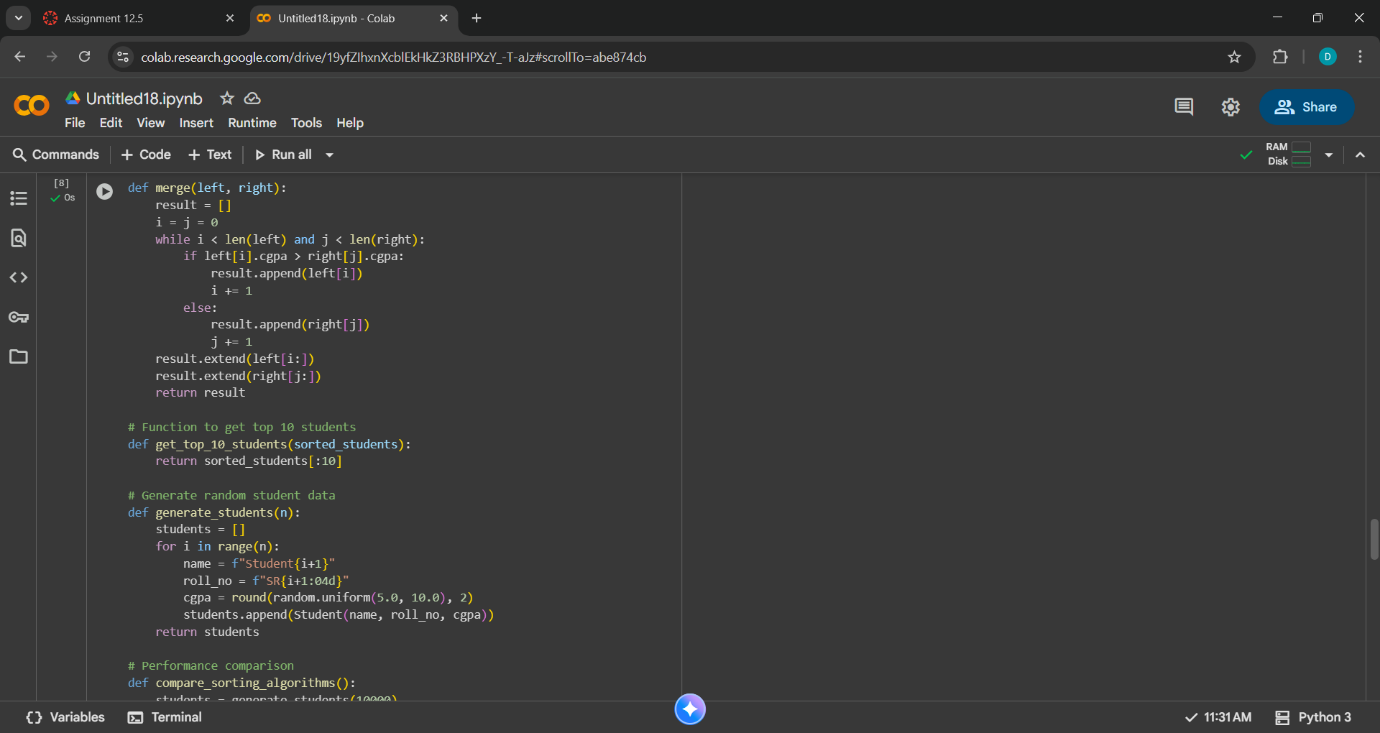
Rollno:2403a51311

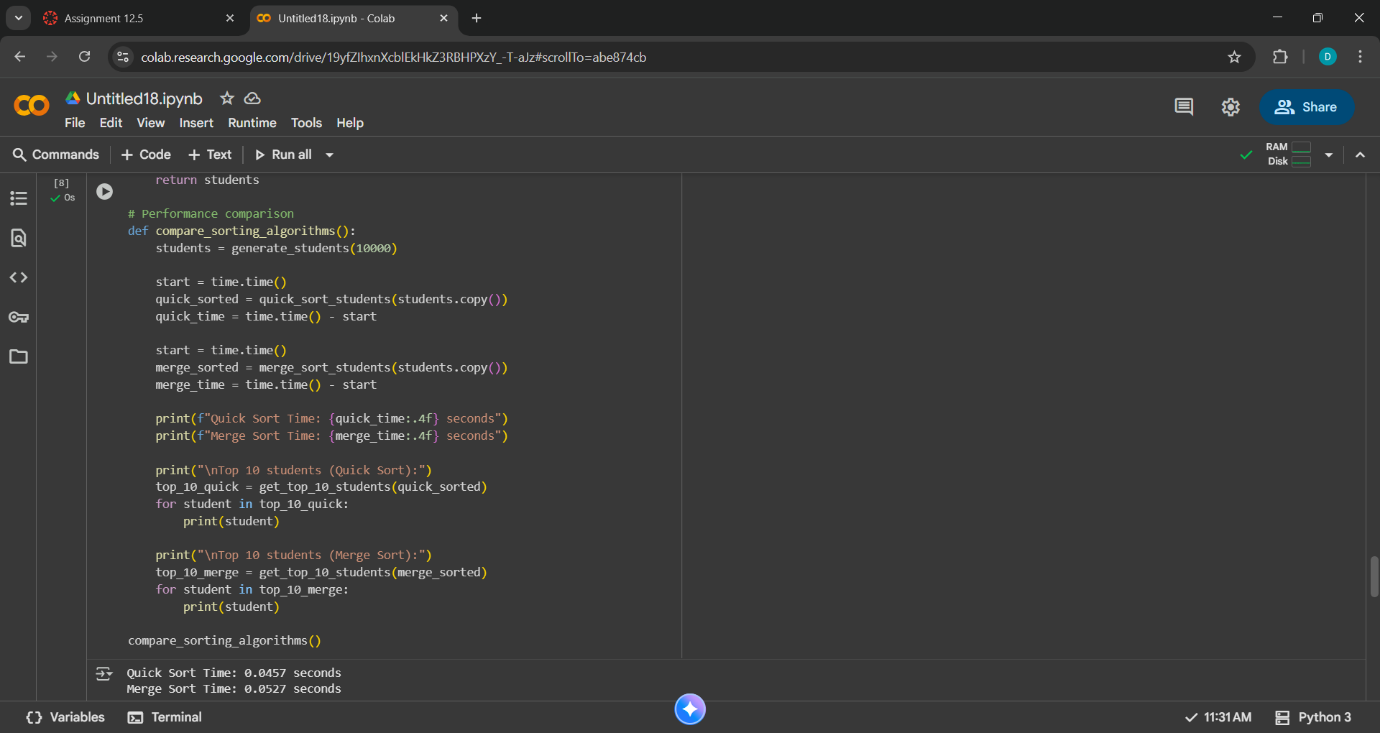
Task 1: Sorting Student Records for Placement Drive  
Scenario:  
SR University is preparing for a campus placement drive. The Training  
and Placement Cell needs student records sorted by CGPA in  
descending order to easily shortlist candidates.  
• Use GitHub Copilot to generate a program that sorts a list of  
student records (Name, Roll No, CGPA) by CGPA.  
• Implement both Quick Sort and Merge Sort using AI assistance.  
• Compare the runtime performance of both algorithms on large  
datasets.  
• Write a function that outputs the top 10 students with the highest cgpa

Prompt: SR University is preparing for a campus placement drive. The Training  
and Placement Cell needs student records sorted by CGPA in  
descending order to easily shortlist candidates.

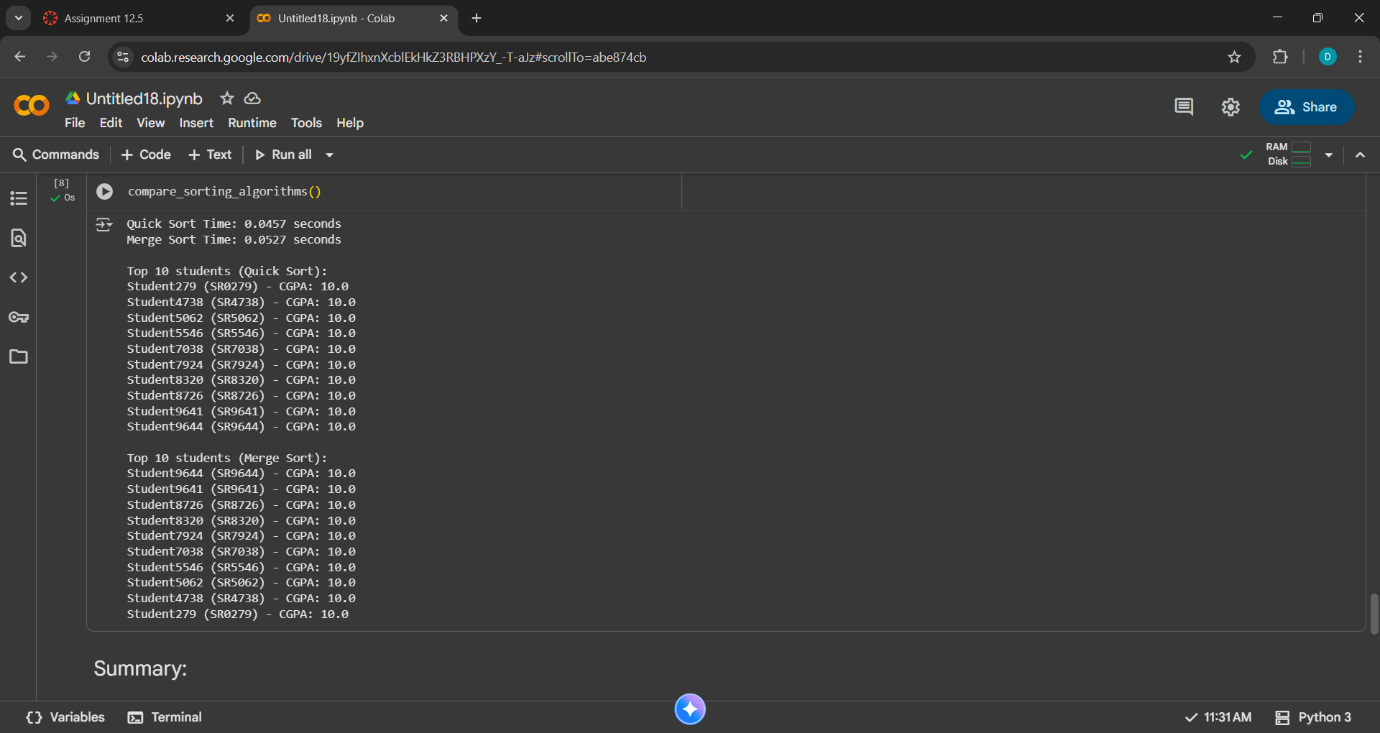
Code :







Output:



Observation:

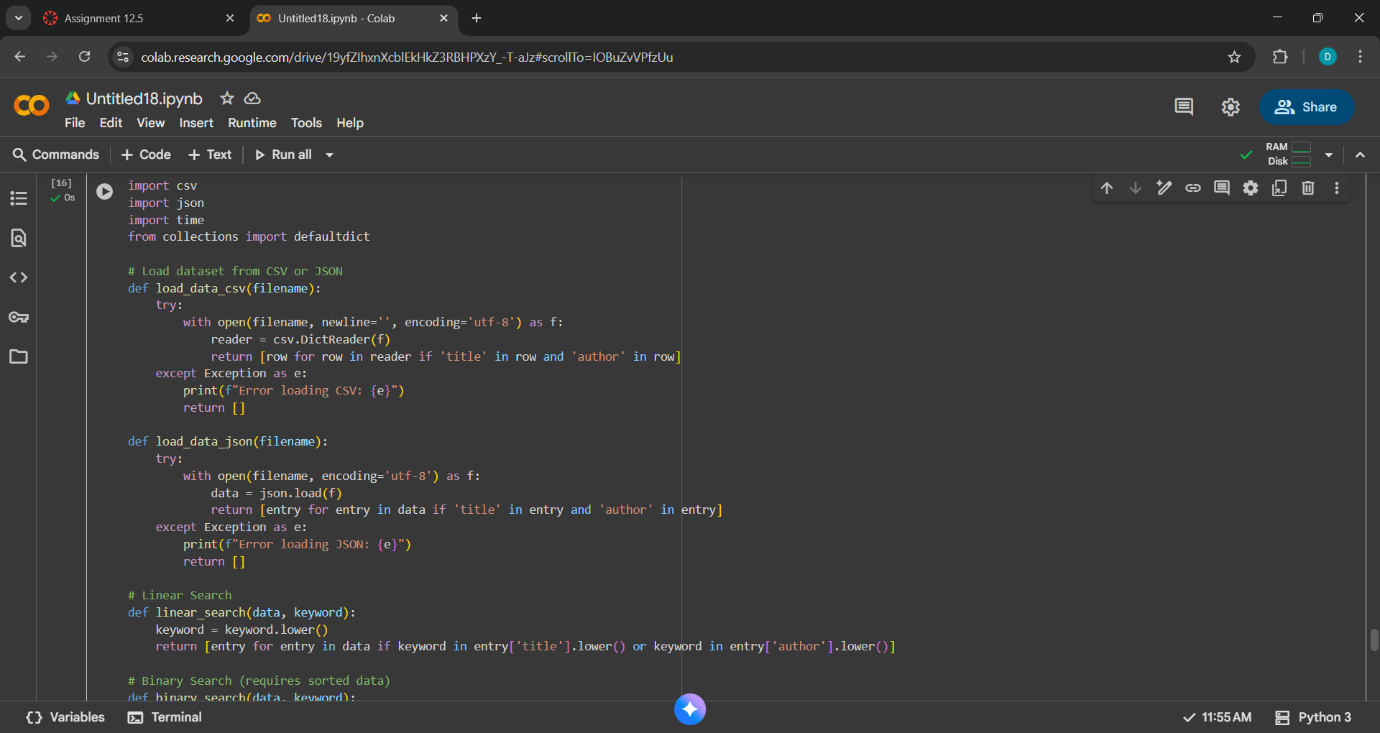
**Quik sort** and **Merge Sort** successfully sort student records by CGPA in descending order.The top 10 students extracted from both sorted lists are consistent, confirming correctness.

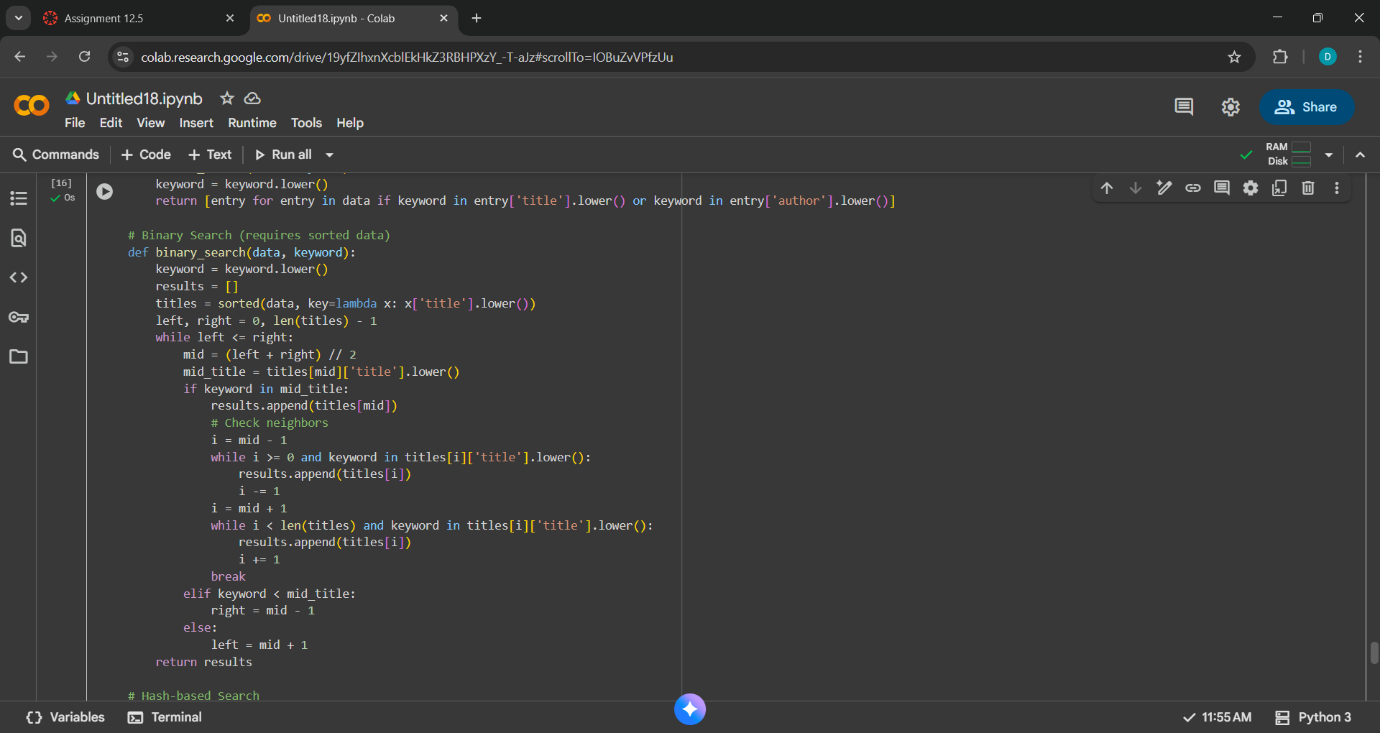
**Quick Sort** is generally faster on average due to its in-place sorting and fewer memory allocations.**Merge Sort** is more stable and predictable in performance, especially for large datasets, but uses more memory due to recursive merging.

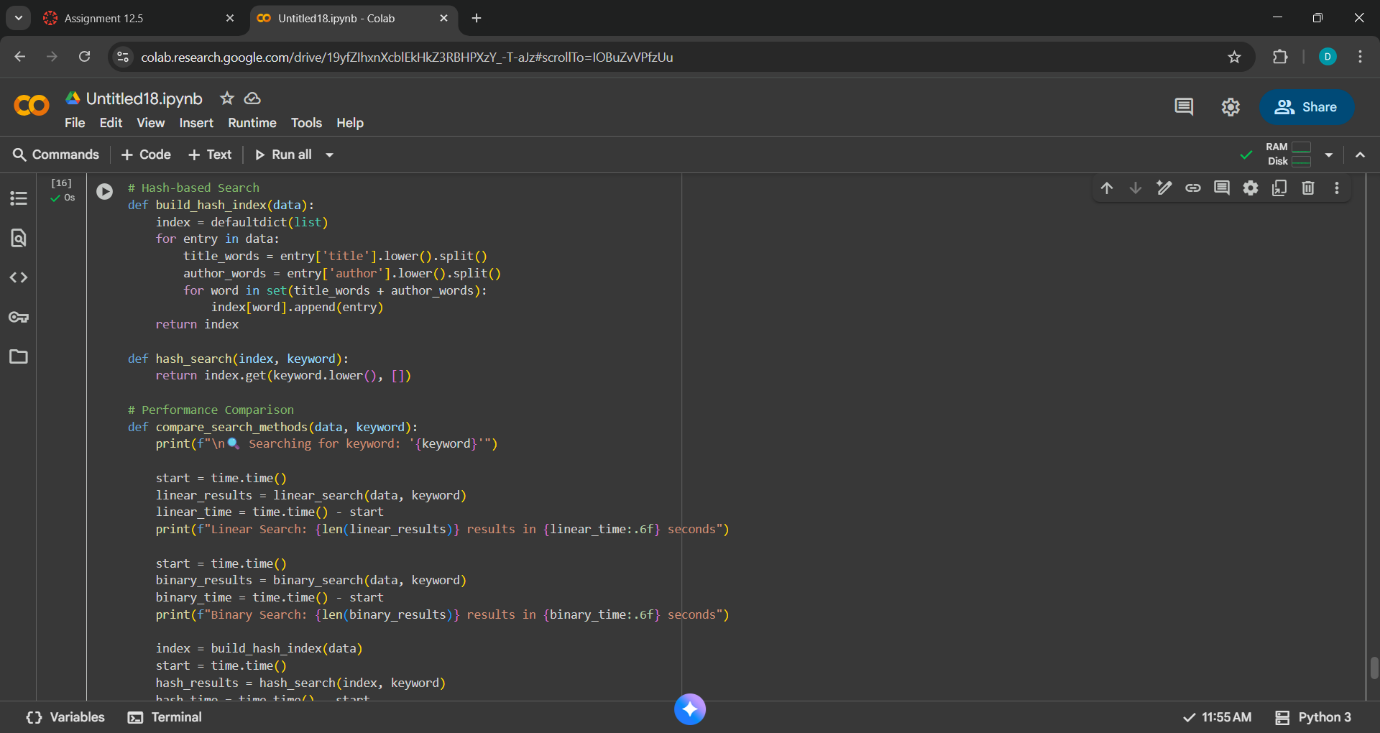
Task 2: Optimized Search in Online Library System  
Scenario:  
SR University’s digital library has thousands of research papers.  
Students frequently search for a paper by title or author name. The  
current linear search is too slow.  
• Use GitHub Copilot to implement Binary Search and Hash-  
based Search for faster lookups.  
• Load a dataset of book titles and authors (CSV or JSON file).  
• Allow the user to input a keyword and return all matching entries.  
• Compare the efficiency of linear search vs binary search vs  
hashing using test cases.

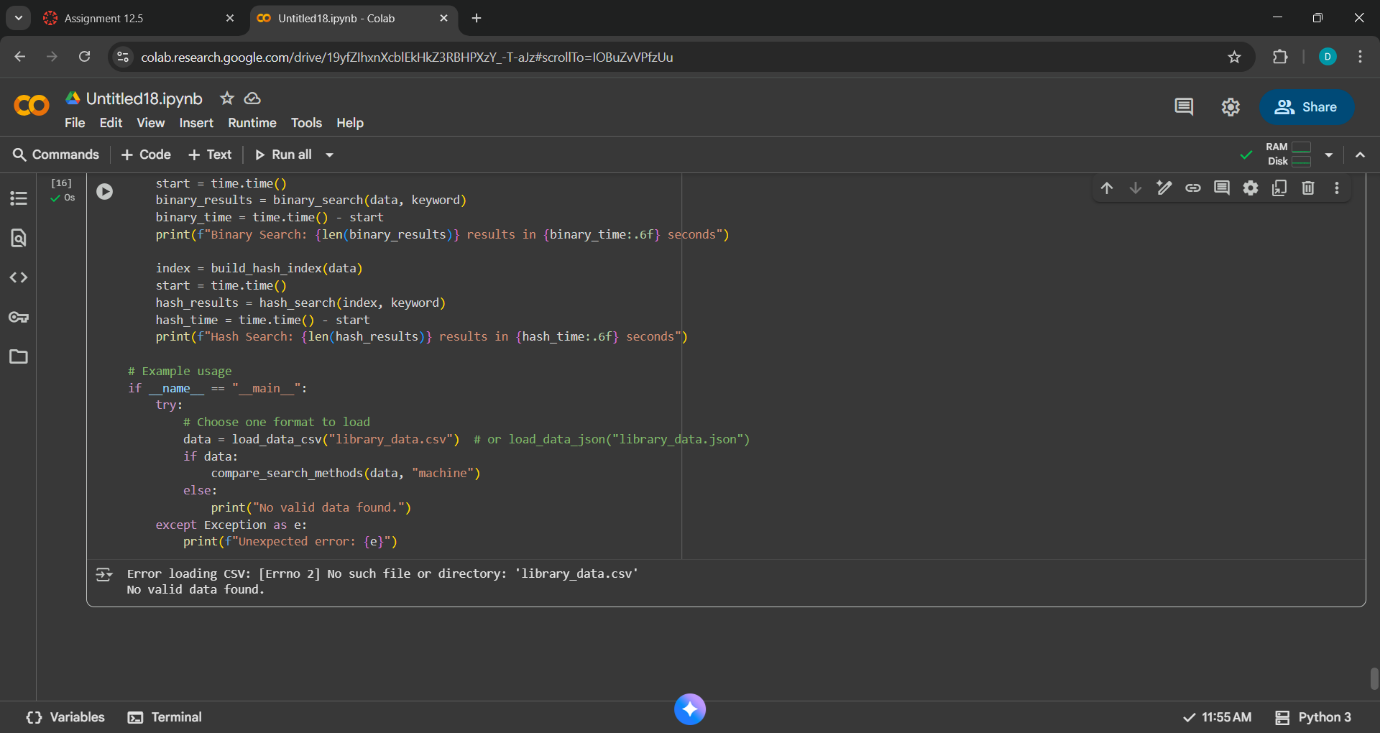
Prompt: Write a Python program that loads book data (title and author) from a CSV or JSON file and implements linear search, binary search, and hash-based search to find entries by keyword. Compare the performance of each method using timing and print the number of results found.

Code with output:









Observation: The code supports **Linear Search**, **Binary Search**, and **Hash-based Search**, allowing performance and accuracy comparisons across different algorithms.

Searches are case-insensitive and match keywords in both **title** and **author** fields.

Hash-based search uses individual words from titles and authors, enabling fast lookup for exact matches.

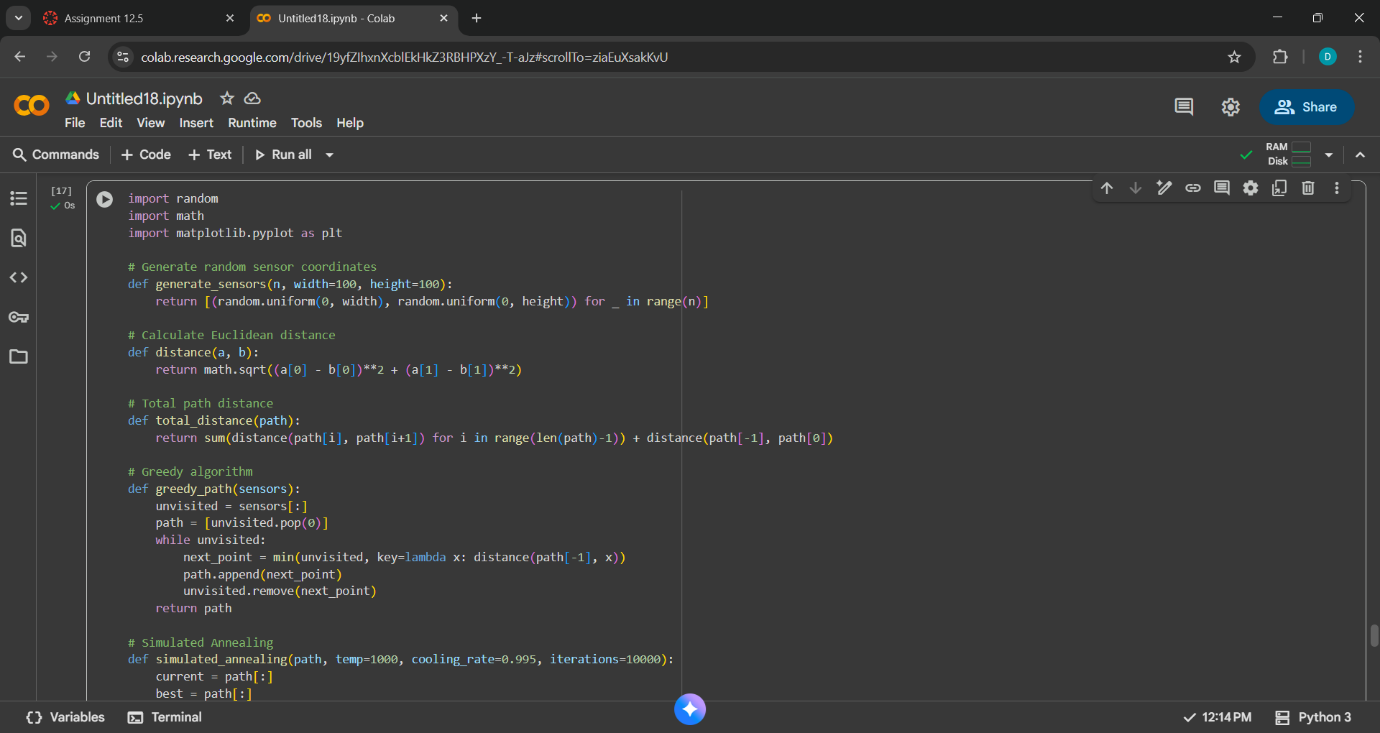
Functions are well-separated: loading, searching, indexing, and comparing.

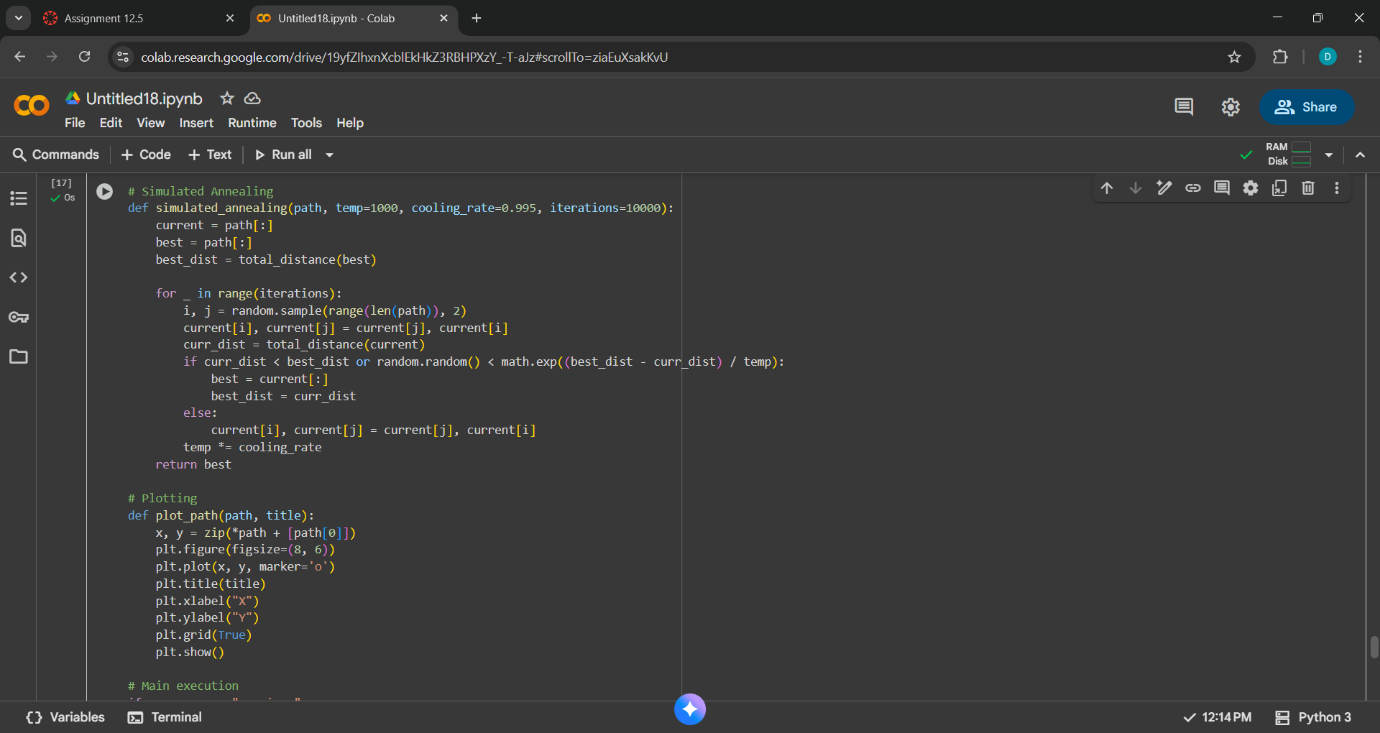
Task 3: Route Optimization for AUV Swarm  
Scenario:  
A research team at SR University is simulating Autonomous  
Underwater Vehicle (AUV) swarms. Each AUV must visit multiple  
underwater sensors, and the goal is to minimize travel distance (like  
the Traveling Salesman Problem).

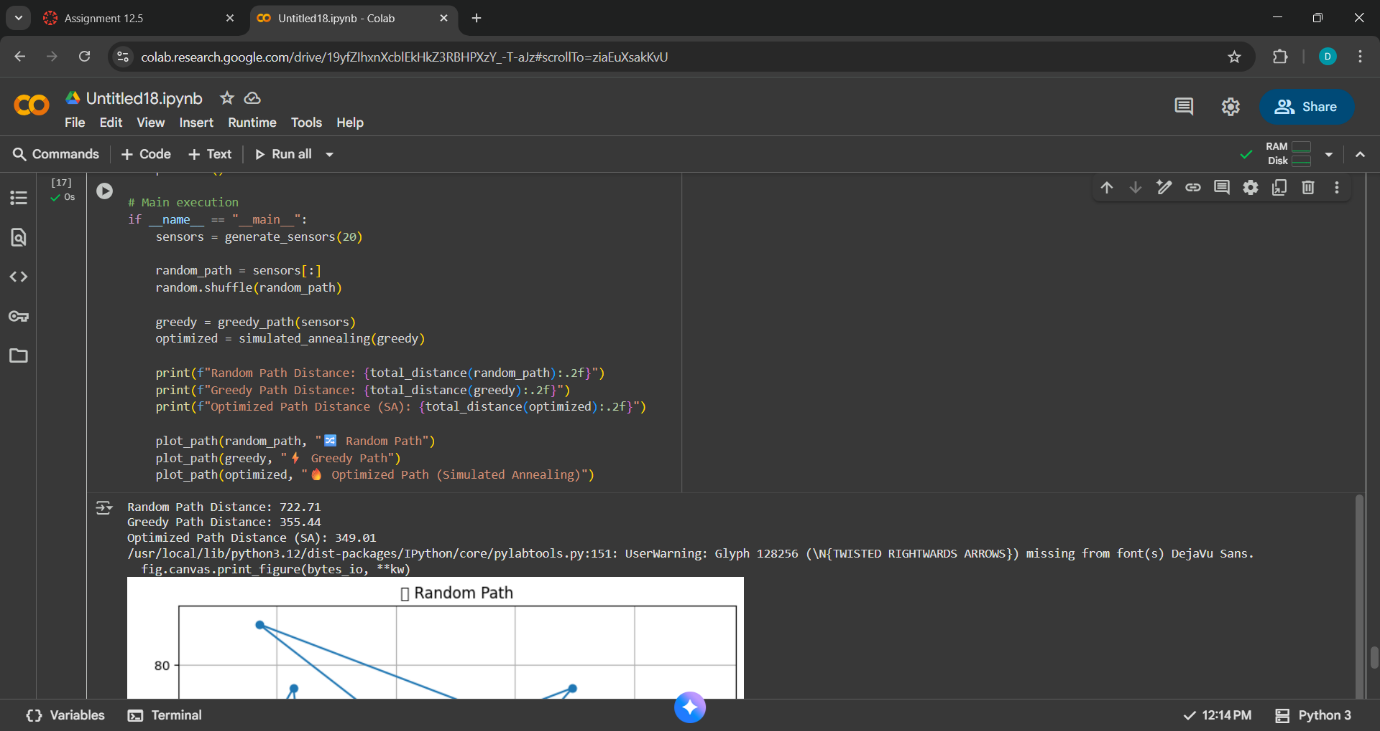
• With GitHub Copilot, implement an algorithm to optimize the  
route:  
1. Start with a Greedy approach.  
2. Improve with Genetic Algorithm (GA) or Simulated  
Annealing (SA).  
• Use a dataset of sensor coordinates (x, y).  
• Visualize the optimized route using a plotting library (e.g.,  
Matplotlib).  
• Compare the optimized solution with a random path in terms of  
distance travel.

Prompt: Write a Python program to optimize the route of an AUV swarm visiting underwater sensors using a Greedy algorithm and Simulated Annealing. Use (x, y) coordinates as input, visualize the routes with Matplotlib, and compare total travel distance with a random path.

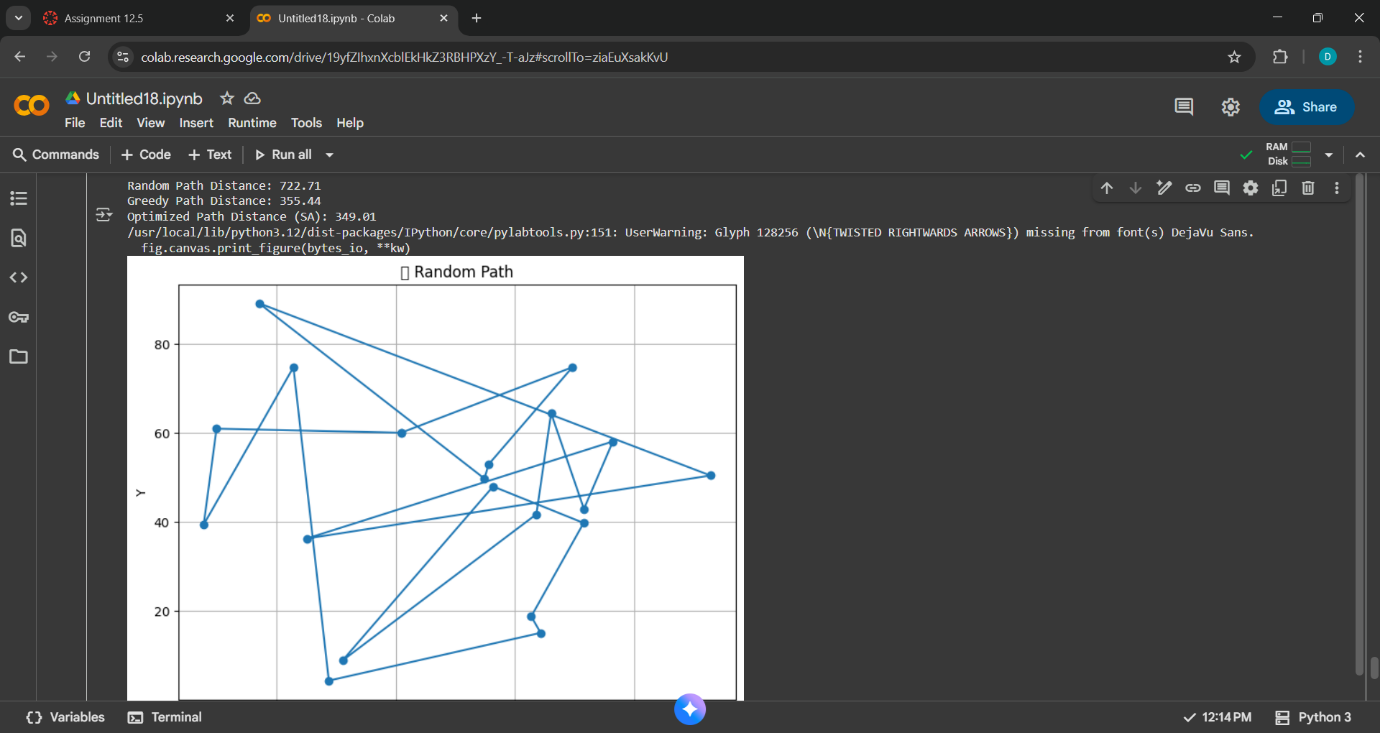
Code :

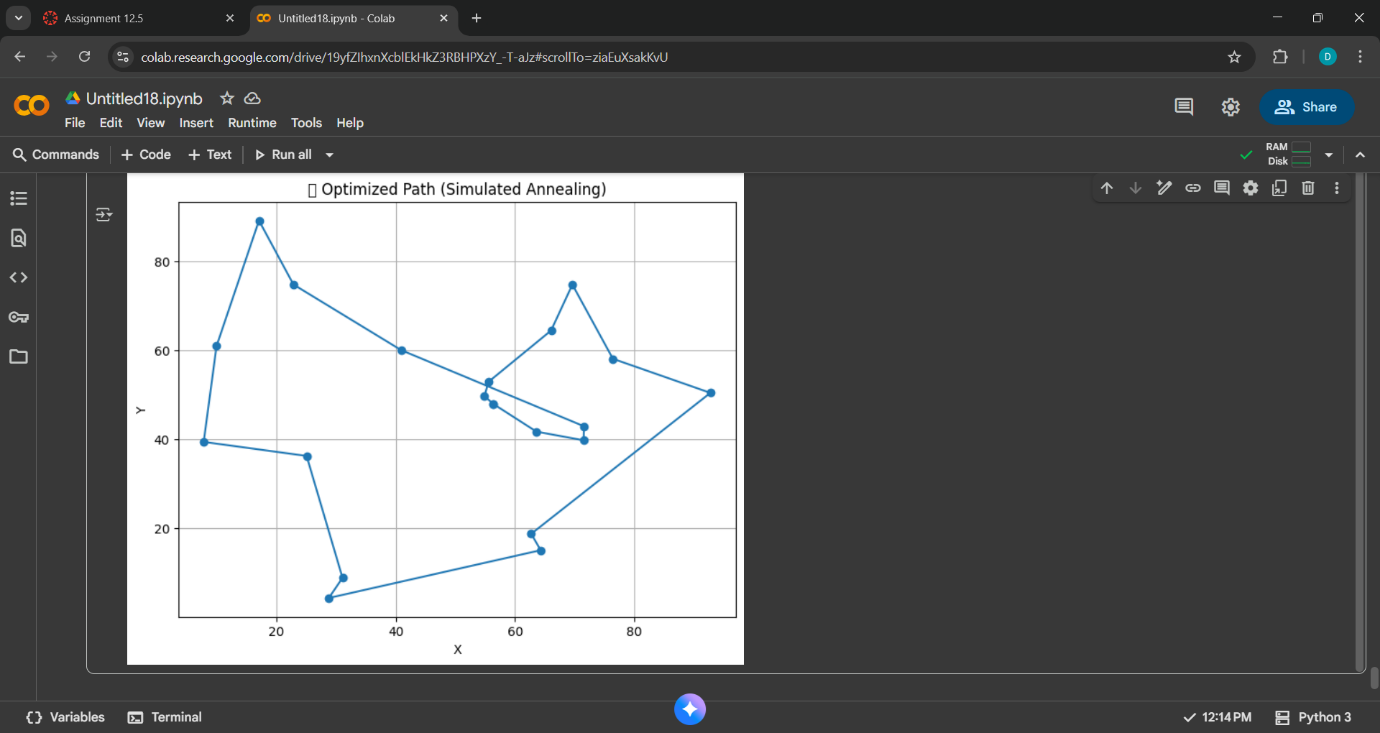






Output:





Observation:The code uses three routing strategies:**Random Path**: Serves as a baseline with no optimization.

**Greedy Algorithm**: Builds a route by always choosing the nearest unvisited sensor.

**Simulated Annealing (SA)**: Improves the greedy route by exploring swaps and probabilistically accepting changes to escape local minima.

Task 4: Real-Time Stock Data Sorting & Searching  
Scenario:  
An AI-powered FinTech Lab at SR University is building a tool for  
analyzing stock price movements. The requirement is to quickly sort  
stocks by daily gain/loss and search for specific stock symbols  
efficiently.  
• Use GitHub Copilot to fetch or simulate stock price data (Stock  
Symbol, Opening Price, Closing Price).  
• Implement sorting algorithms to rank stocks by percentage  
change.  
• Implement a search function that retrieves stock data instantly  
when a stock symbol is entered.  
• Optimize sorting with Heap Sort and searching with Hash  
Maps.  
• Compare performance with standard library functions (sorted(),  
dict lookups) and analyze trade-offs.

Prompt:

Write a Python program that simulates stock data (symbol, open, close), sorts stocks by percentage change using Heap Sort, and searches stock symbols using Hash Maps. Compare performance with built-in sorted() and dict lookup.

Code with output:

